ARP Request Support

Make it work the first time

Wed, May 21, 1997

When using Internet Protocols (IP), one must use the Address Resolution Protocol (ARP) to obtain a network physical address given a target IP address. Replies from ARP requests are cached in an ARP table so that ARP requests don't have to be performed too often. After some time, an ARP table entry is normally flushed, so that the next time it is necessary to send to that IP address, a new ARP request must be used. Although an IP implementation may hold back datagrams to be sent to a target IP address until it has obtained the hardware address, it is not required to do so. In many implementations, when faced with sending a datagram to a target IP address on the local network, the system sends an ARP request *instead* of the message, hoping that a high level retry will subsequently find a physical network address in the cache so that the datagram can be delivered. The IRM implementation of IP was designed to work in just this way. This note describes a method of implementing support for delaying a datagram until an ARP reply has been received so that ARP requests will occur automatically as needed without omitting the first datagram.

IRM Networking Background

IRM network support has had a long history. Network software organization in the preceding VME local station implementation even preceded IP support. It is designed around the communication of messages, as distinct from frames or datagrams. In both Classic and Acnet protocols, multiple messages that target the same node are concatenated into single frames or datagrams as can fit. (In some cases, messages cannot be combined, say in the case that a message is really the entire contents of a UDP datagram.) The concatenation of messages is handled by low level network code that is invoked after a network queue, one per network, has been loaded with pointers to network message blocks. Concatenation works with consecutively queued messages that target the same node and, in the case of UDP, the same node/port, or socket. Much of a front end's network communication consists of replies to data requests. A linked list of all outstanding data requests is maintained in an order such that multiple requests from the same node are grouped together. This scheme increases the likelihood that reply messages to multiple data requests from the same node that are due at the same time will be combined into a common frame or datagram for delivery. Often a number of IRMs are logically connected through a data server node, so that the chance of having multiple requests stemming from a common node is even more likely. Concatenation, of course, improves the efficiency of network utilization and is therefore deemed a "good thing."

The support for queuing all network messages is handled by a common routine called OUTPQX. It has the job of placing the pointer to a network message block into an OUTPQ ("output pointer queue") according to the target network. The target network is determined by the destination node# word found in the message block. Ranges of

Node# (hex)	*Network*
000x–00Ex	token ring via Acnet logical node table
00Fx	multicast raw
01xx–03xx	token ring raw
04xx	token ring raw
05xx–07xx	token ring raw *or* ethernet IP via DNS
08xx	token ring raw via token ring/ethernet bridge
09xx–10xx	token ring IP or ethernet IP
09Fx	multicast IP
7Axx	arcnet raw
8xxx	ethernet raw
6xxx	token ring UDP socket
Exxx	ethernet UDP socket

The range 05xx–07xx has a special significance. This range may imply use of token ring or ethernet, and it may be raw or IP. Node#s in this range will be sent via IP if the local node's global "broadcast" node# is in the range 09xx. (The "broadcast" node# is usually a multicast node# this is used to target requests that must be fulfilled by contributions from more than one other node, or to target Classic protocol device name lookup requests.) To get the IP address, the IP Node Address Table IPNAT is consulted that holds cached IP addresses that are derived from the local Domain Name Server via the local application DNSQ. All current IRMs are configured with a "broadcast" node of 09Fx. The choice of ethernet or token ring is made depending on CPU board and whether IP is to be used. On an MVME162 board, ethernet is always used for IP; ethernet is also used for non-IP (raw) when there is no token ring interface present. In practice, IRMs use ethernet on IP. MVME133 board-based systems use token ring, either IP or raw. Only a 162 board system can use both token ring and ethernet, and it will always use ethernet for IP.

IP communications is based upon a socket, which specifies an IP address and a UDP port#. (ICMP and IGMP effectively use a zero port#.) The low 12 bits of the socket node# in the above table refer to an 8-bit index into the ARP table and a 4-bit port# index. Each ARP table entry in active use has an associated port# block that can contain up to 15 active port#s from that node. In this way, a single word used as a target node# encodes a UDP socket. There are 254 ARP table entries available for sockets in active use, each of which can deal with up to 15 active UDP port#s.

New scheme for handling ARP requests

The networking support in IRMs has always been "in a hurry." An IRM is a front end that adheres to real-time performance. The notion of holding up network communications with other nodes while awaiting an ARP reply is not consistent with

realtime functionality. As a rule to be followed, it's "ok" to hold up messages that are to be sent to a target node for which an ARP request must be sent, but it's not "ok" to hold up messages to be sent to other nodes. The new scheme honors this rule.

New code was added to the OUTPQX routine to detect the case that an ARP request will frustrate delivery of a message using IP, if the message were allowed to pass on to the network software that builds frames. In such a case, an ARP request message is broadcast to the network, a ptr to the message block is queued in a data structure linked with the corresponding ARP table entry, and success is returned to the caller, implying that the message block has been "queued to the network." When the ARP reply is received, often within a few milliseconds, and the hardware address found therein is deposited into the corresponding ARP table entry, all the queued message block ptrs are passed through OUTPQX again, this time with assurance that they will really be queued to the network. If no ARP reply is forthcoming, after a couple of seconds, the message ptrs are passed through OUTPQX in such a way that they are queued to the appropriate OUTPQ, but they are marked so that the lower level network frame-building software will ignore them. As a result, they can still be handled by the Queue Monitor task, which has the responsibility of freeing blocks that are no longer needed following completion of transmission, or of marking them no longer busy so they are available for subsequent retry use.

An ARP queue block that is allocated to house the queued message block ptrs is large enough to hold 13 ptrs. If more are needed, another ARP queue block is allocated and linked to the first, so that there is no real limit to the number of messages that can be queued awaiting an ARP reply.

To implement this new ARP queue support, changes were made to OUTPQX as described above, to the IPARP suite of routines that support access to the ARP table, and to the SNAP Task, which handles ARP replies.